Sample *Matplotlib* Plotting Recipes

1 Imports

The most commonly used imports for creating plots with *Python*.

```
import math                         # usually only math or numpy, not both
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rc           # less frequent
```

2 Create data points for plotting

Use *NumPy* to create an array of x values and then use x to calculate y values for plotting. Standard *Python* may also be used, but *NumPy* is much more efficient most of the time.

```
x = np.linspace(-np.pi*2, np.pi*2, 200)
y1 = x*np.cos(x)
y2 = x*np.sin(x)
```

Matplotlib offers a number of styles for creating plots if you don't want to start with and tweak the default style. However, the default style is sufficient most of the time, especially for technical documents.

```
print(plt.style.available)
plt.style.use('default')
```

```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery',
 ↪ '_mpl-gallery-nogrid', 'bmh', 'classic', 'dark_background', 'fast',
 ↪ 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn-v0_8',
 ↪ 'seaborn-v0_8-bright', 'seaborn-v0_8-colorblind',
 ↪ 'seaborn-v0_8-dark', 'seaborn-v0_8-dark-palette',
 ↪ 'seaborn-v0_8-darkgrid', 'seaborn-v0_8-deep', 'seaborn-v0_8-muted',
 ↪ 'seaborn-v0_8-notebook', 'seaborn-v0_8-paper',
 ↪ 'seaborn-v0_8-pastel', 'seaborn-v0_8-poster', 'seaborn-v0_8-talk',
 ↪ 'seaborn-v0_8-ticks', 'seaborn-v0_8-white',
 ↪ 'seaborn-v0_8-whitegrid', 'tableau-colorblind10']
```

3 Using TEX for text objects in plots

You can change *Matplotlib's* settings to use the TEX system and add fancy formatting to text items in plots; such as Greek characters and formulas with subscripts and superscripts. The default setting is to have TEX turned off.

```
# include the following line to turn TeX on
rc('text', usetex=True)
```

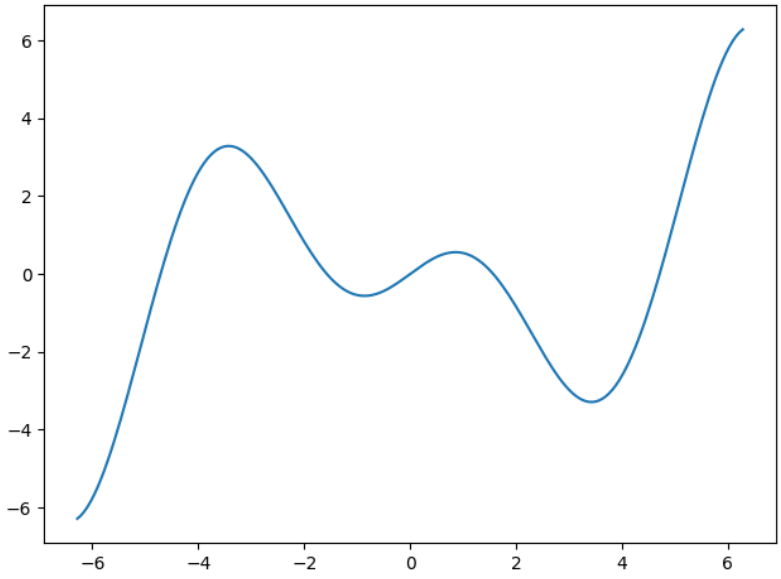
4 *Matplotlib* object-oriented plotting

Following are a number of common plotting "recipes" that use *Matplotlib's* object-oriented approach. The same plot data is used a number of times with changes to the plot settings each time so the user can see what affect each change has on the plot.

4.1 Simple Plots

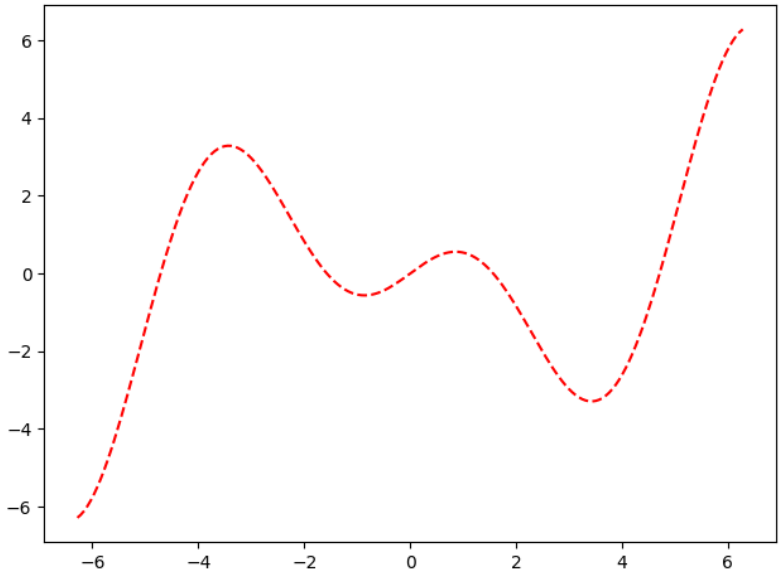
Create a figure and axes using the `plt.subplots()` function then plot to the axes and show the plot.

```
rc('text', usetex=False)     # turn off TeX for now
fig, ax = plt.subplots()
ax.plot(x, y1)
plt.show()
```



Use a formatting string inside the `plot()` function to change the color and line type of the plotted curve. The `'r--'` option is a shorthand notation for setting the color, line style, and/or marker shape. Each is optional and may either be included or omitted. A separate document lists all of the available options.

```
fig, ax = plt.subplots()
ax.plot(x, y1, 'r--')
plt.show()
```



The following code block creates a "plot" of all of the common marker indicators and line styles. *Matplotlib* automatically cycles through different colors for different plots in the same space.

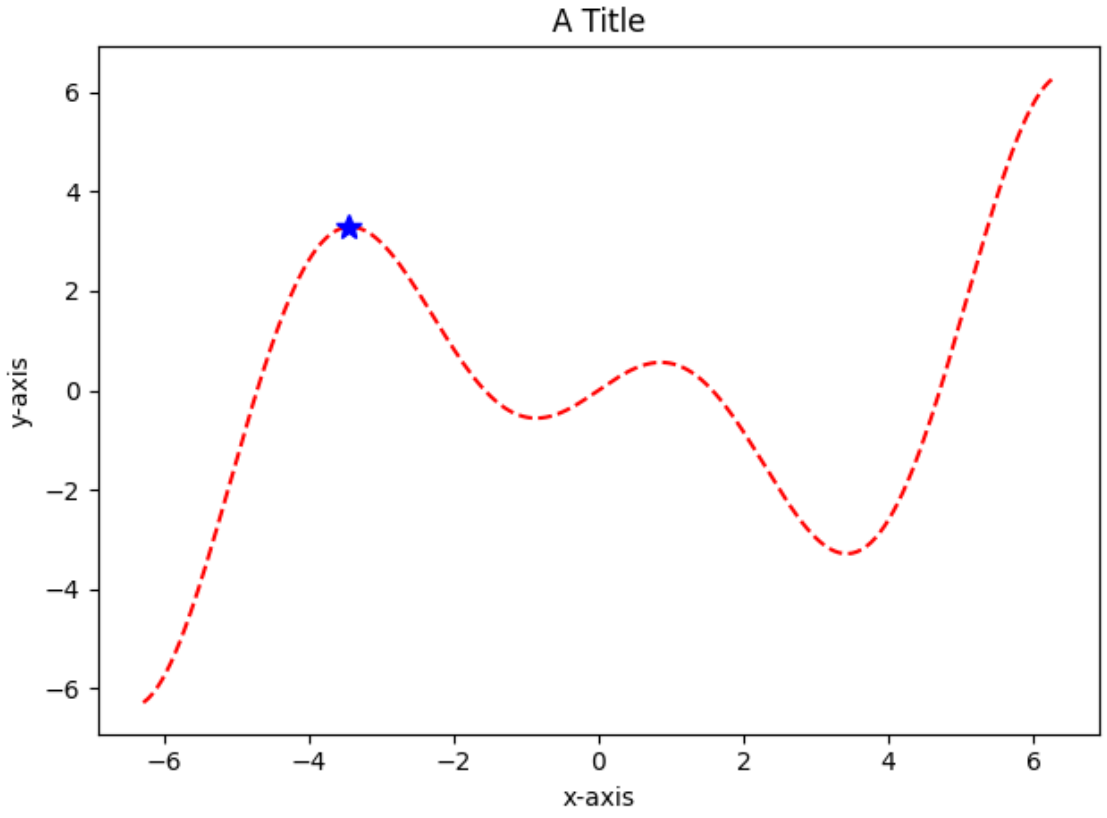
```
fig, ax = plt.subplots(figsize=(6,1))
for i,item in enumerate(['-','--','-.',':']):
    ax.plot([0, 17], [2*i+2, 2*i+2], item, lw=2.5)
for i,item in enumerate('.xX+PodDs^v<>hH*p8'):
    ax.plot(i, 0, item, ms=8)
ax.set_axis_off()
ax.set_ylim(-1, 9)
plt.show()
```

5 Adding Features

5.1 Adding Labels, Limits, and Grids

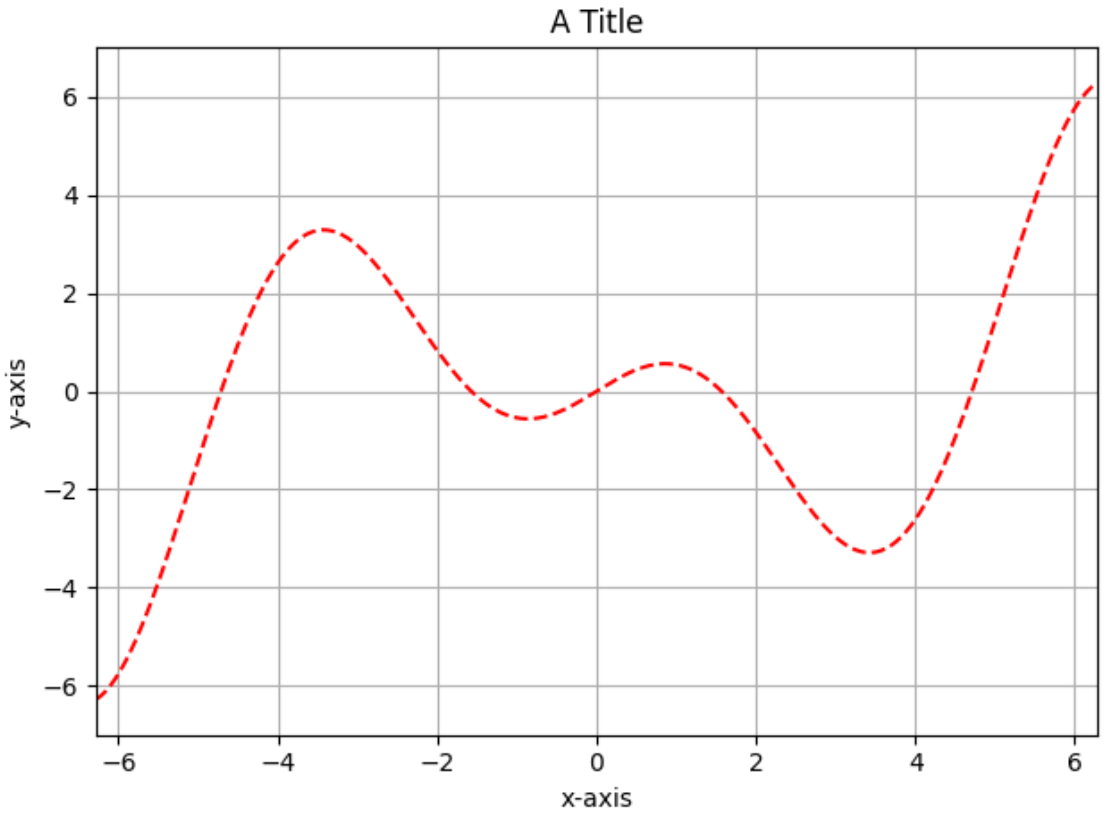
A single point may be plotted by simply using the x and y values of the point as arguments in the `plot()` function. It this example the maximum y value and its corresponding x value are plotted with a blue star along with the original plot. This plot also adds a title on the top and axes labels.

```
fig, ax = plt.subplots()
ax.plot(x, y1, 'r--')
ax.set_title('A Title')      # add a title to 'ax'
ax.set_xlabel('x-axis')      # add an x-axis label to 'ax'
ax.set_ylabel('y-axis')      # add a y-axis label to 'ax'

ymax = np.max(y1[:100])
xmax = x[np.argmax(y1[:100])]

ax.plot(xmax, ymax, 'b*', markersize=10)    # plotting max y point
plt.show()
```



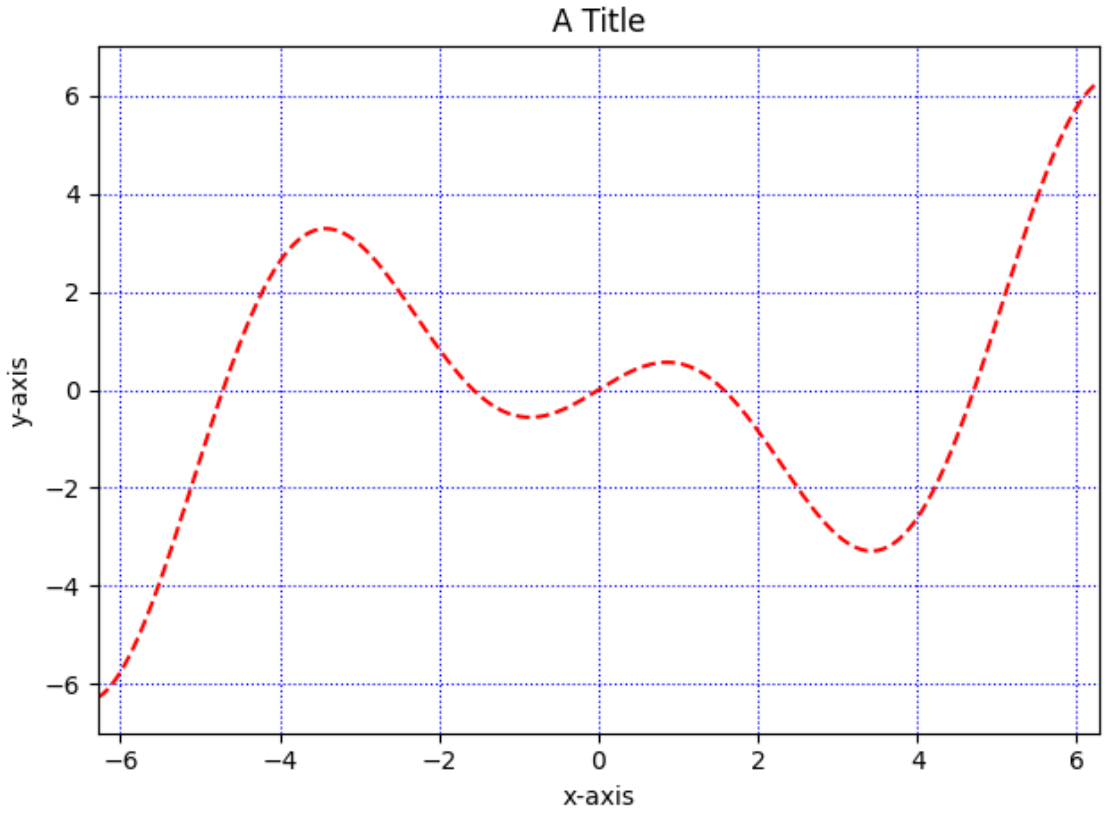
The following plot includes a grid and explicitly sets the axes limits using `ax.axis()`. In this case, keyword arguments for each parameter are being used in the `ax.axis()` function. Notice that the limits can be determined via calculated expressions and are not limited to integer values.

```
fig, ax = plt.subplots()
ax.plot(x, y1, 'r--')
ax.set_title('A Title')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
# set display limits of each axis
ax.axis(xmin=-2*np.pi, xmax=2*np.pi, ymin=-7, ymax=7)
ax.grid(True)  # turn grid on
plt.show()
```



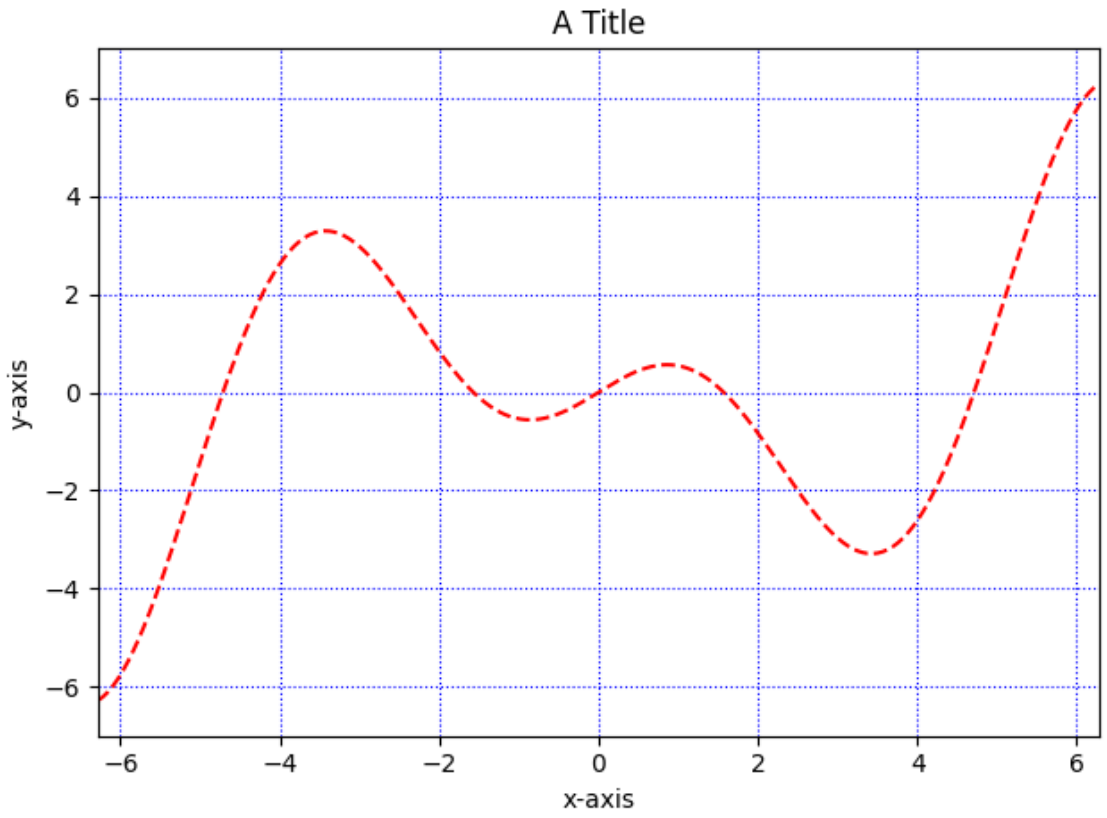
The results of this next plot are essentially the same as the previous plot. However, here a list of axes limit parameters is used as an argument in `ax.axis()`. The grid color is tweaked to use a blue, dotted line as well.

```
fig, ax = plt.subplots()
ax.plot(x, y1, 'r--')
ax.set_title('A Title')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
# ax.axis order is xmin, xmax, ymin, ymax
ax.axis([-2*np.pi, 2*np.pi, -7, 7])
# turn grid on with linestyle = dotted and color = blue
ax.grid(True, ls=':', c='blue')
plt.show()
```



The plot axes limits can also be set separately for each axis using `ax.set_xlim()` and `ax.set_ylim()`. The following plot uses these functions to set the viewing limits of the plot instead of the `ax.axis()` function.

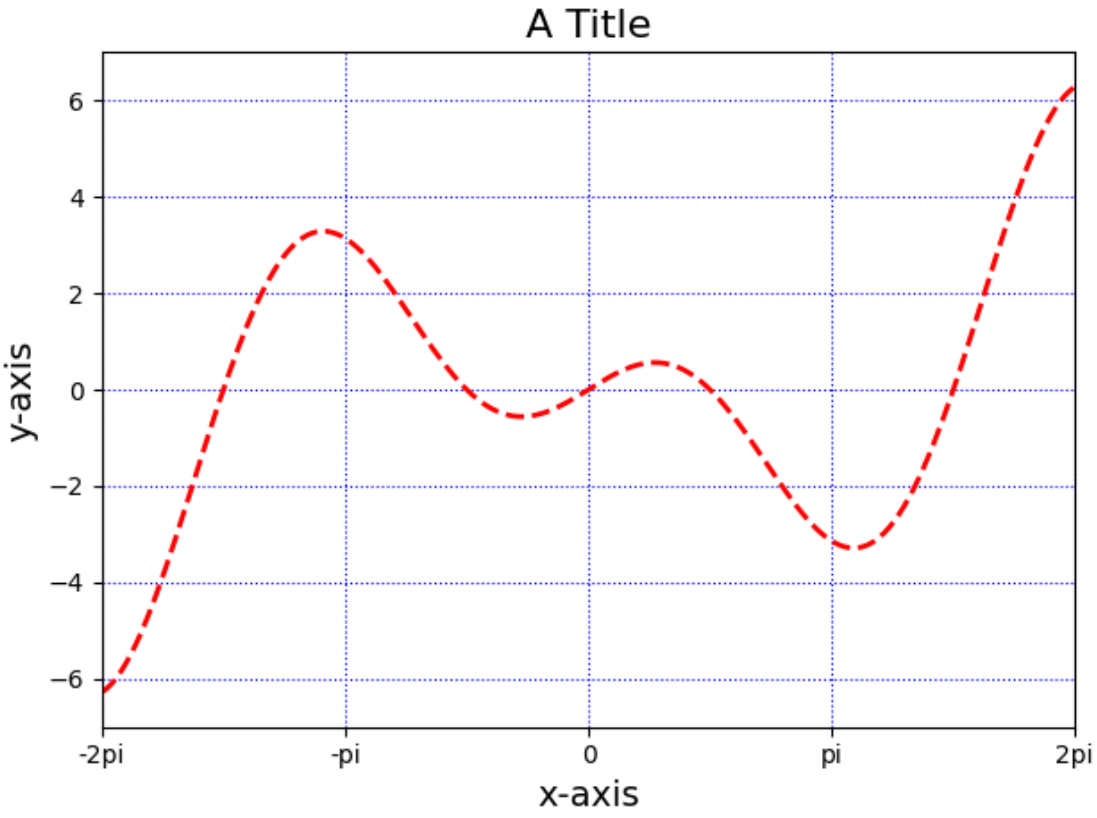
```
fig, ax = plt.subplots()
ax.plot(x, y1, 'r--')
ax.set_title('A Title')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_xlim(-2*np.pi, 2*np.pi)  # just set x-axis limits
ax.set_ylim(-7, 7)              # just set y-axis limits
ax.grid(True, ls=':', c='blue')
plt.show()
```



5.2 Font Sizes and Custom Tick Labels

Sometimes the text in the titles and/or axes labels is not large enough. Add the keyword argument `fontsize=x` to set a non-default font height of `x` points. The line width of this plot has also been made wider by including the keyword argument `lw=2` in the `plot()` function. Use `ax.set_xticks()` and `ax.set_xticklabels()` to customize the x axis tick mark labels. Here they are changed to multiples of π. Something similar couuld be done for the y axis tick marks as well.

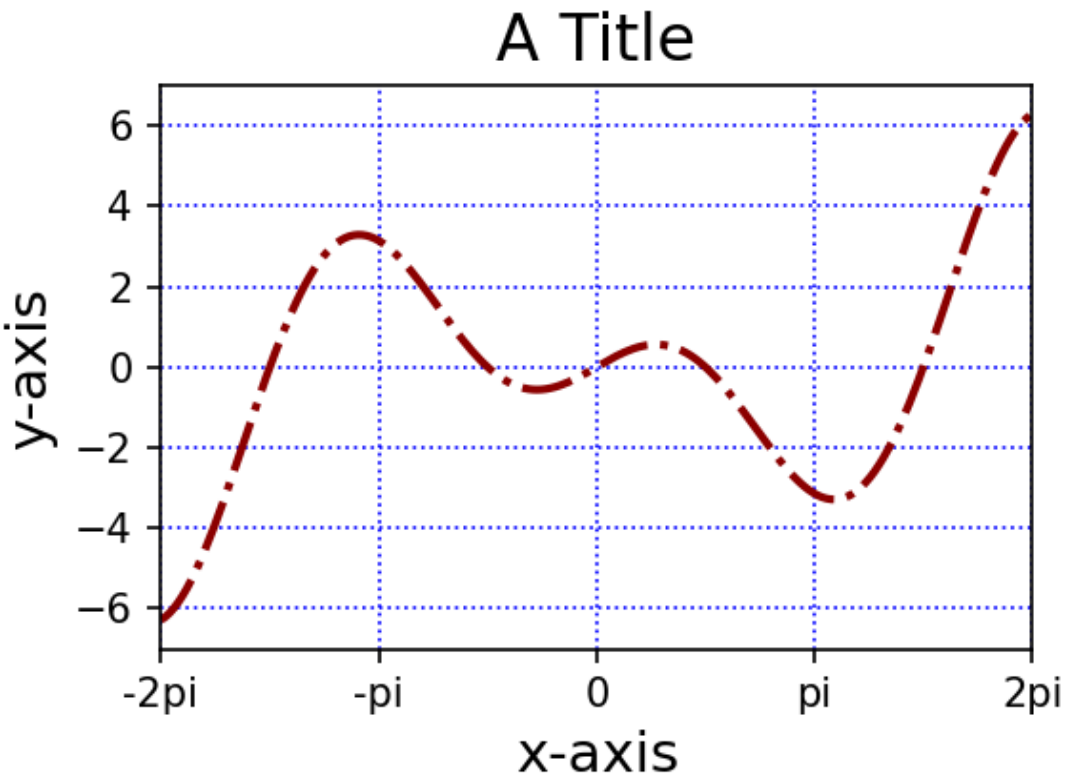
```
fig, ax = plt.subplots()
ax.plot(x, y1, 'r--', lw=2)  # change plot lineweight to 2
ax.set_title('A Title', fontsize=16) # adjust the font size to 16 points
ax.set_xlabel('x-axis', fontsize=14)
ax.set_ylabel('y-axis', fontsize=14)
# set locations of x tick marks
ax.set_xticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi])
# custom labels for x tick marks
ax.set_xticklabels(['-2pi', '-pi', '0', 'pi', '2pi'])
ax.set_xlim(-2*np.pi, 2*np.pi)
ax.set_ylim(-7, 7)
ax.grid(True, ls=':', c='blue')
plt.show()
```



5.3 Changing Figure Size and Pixel Density

The following plot includes the `figsize` and `dpi` keyword arguments in the `subplots()` function to set a custom figure size and pixel density. It also includes an alternate method for changing the plot color and line style by using the keyword arguments `c='darkred'` and `ls='-.'`.

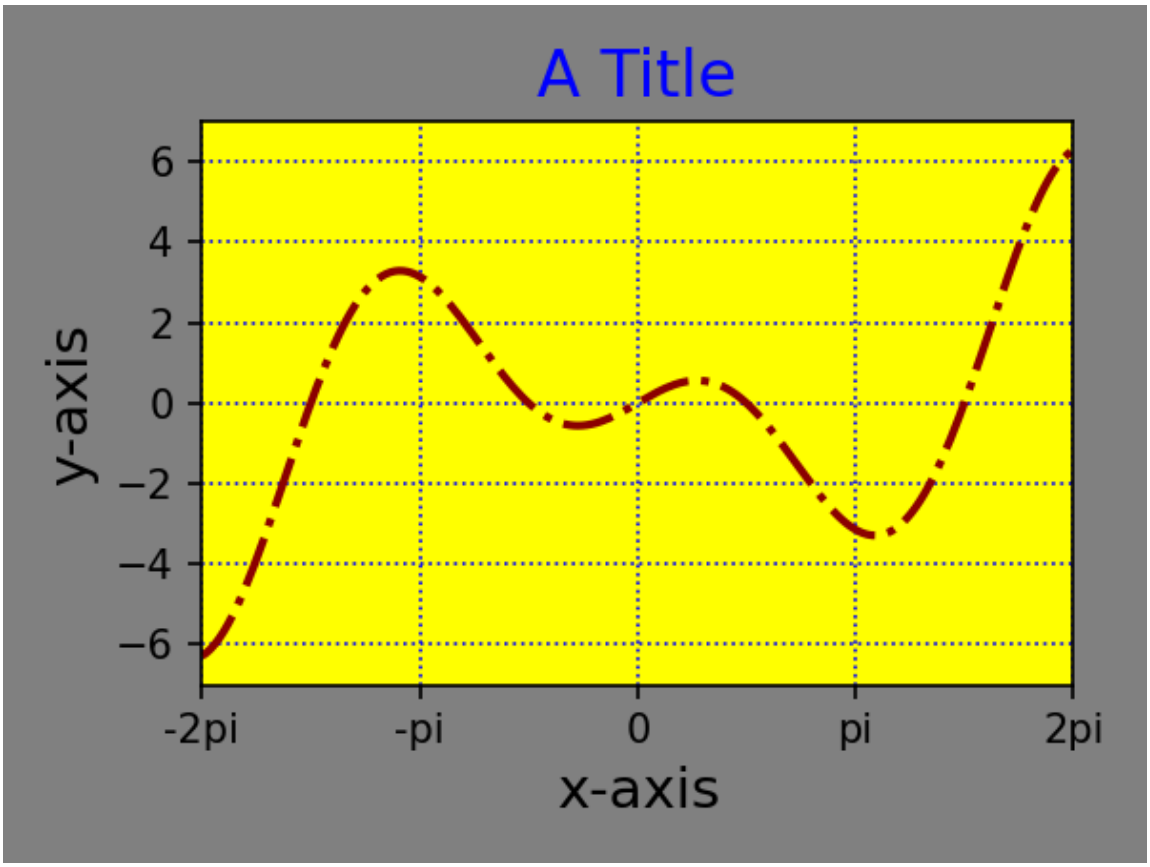
```
fig, ax = plt.subplots(figsize=(4,3), dpi=150)
ax.plot(x, y1, lw=2, c='darkred', ls='-.')
ax.set_title('A Title', fontsize=16)
ax.set_xlabel('x-axis', fontsize=14)
ax.set_ylabel('y-axis', fontsize=14)
ax.set_xticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi])
ax.set_xticklabels(['-2pi', '-pi', '0', 'pi', '2pi'])
ax.set_xlim(-2*np.pi, 2*np.pi)
ax.set_ylim(-7, 7)
ax.grid(True, ls=':', c='blue')
plt.show()
```



5.4 Setting Background Colors

Here the last plot is "embellished" with some rather gaudy color choices by using `ax.set_facecolor()` and `fig.set_facecolor()`. The title text color is also customized by adding the `color='blue'` keyword argument to the title creation function. Note that `c='blue'` and `color='blue'` do the same thing.

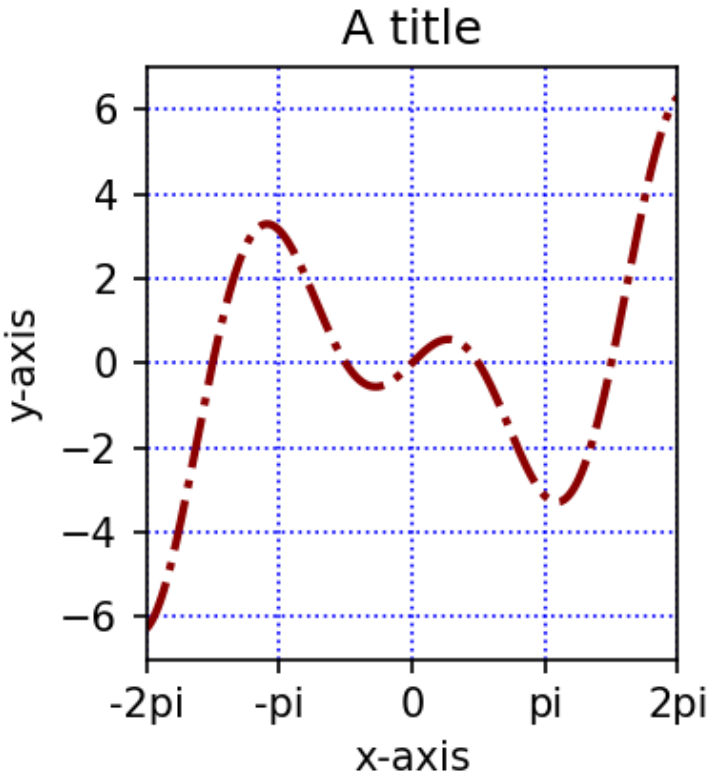
```
fig, ax = plt.subplots(figsize=(4,3), dpi=150)
ax.plot(x, y1, lw=2, c='darkred', ls='-.')
ax.set_facecolor('yellow')   # add a color to the plot area
fig.set_facecolor('grey')    # add a figure color (around plot axes)
ax.set_title('A Title', fontsize=16, color='blue') # change text color
ax.set_xlabel('x-axis', fontsize=14)
ax.set_ylabel('y-axis', fontsize=14)
ax.set_xticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi])
ax.set_xticklabels(['-2pi', '-pi', '0', 'pi', '2pi'])
ax.set_xlim(-2*np.pi, 2*np.pi)
ax.set_ylim(-7, 7)
ax.grid(True, ls=':', c='blue')
plt.show()
```



5.5 Aspect Ratios

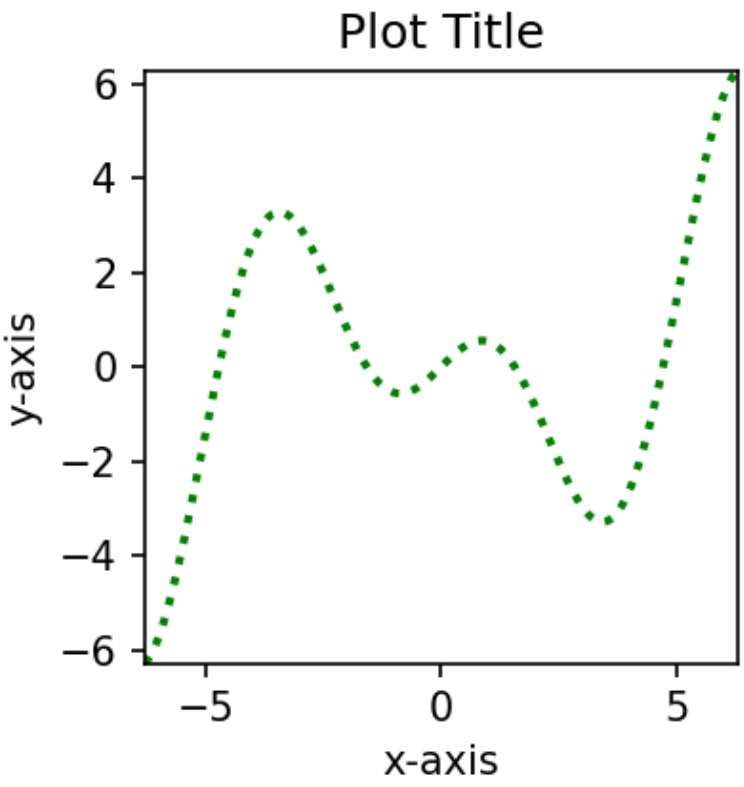
The x and y aspect ratios in the previous plots have not matched each other. If the units for both axes of a plot are the same, it might be nice to make the aspects match. In the following plot the `ax.set()` function is used with the `aspect='equal'` keyword argument to make one unit on the x axis equal one unit on the y axis. It also is setting all of the plot labels at once using a separate `ax.set()` function call.

```
fig, ax = plt.subplots(figsize=(4,3), dpi=150)
ax.plot(x, y1, lw=2, c='darkred', ls='-.')
ax.set(aspect='equal')  # 1 unit on x-axis = 1 unit on y-axis
ax.set(title='A title', xlabel='x-axis', ylabel='y-axis') # all labels at
↪  once
ax.set_xticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi])
ax.set_xticklabels(['-2pi', '-pi', '0', 'pi', '2pi'])
ax.set_xlim(-2*np.pi, 2*np.pi)
ax.set_ylim(-7, 7)
ax.grid(True, ls=':', c='blue')
plt.show()
```



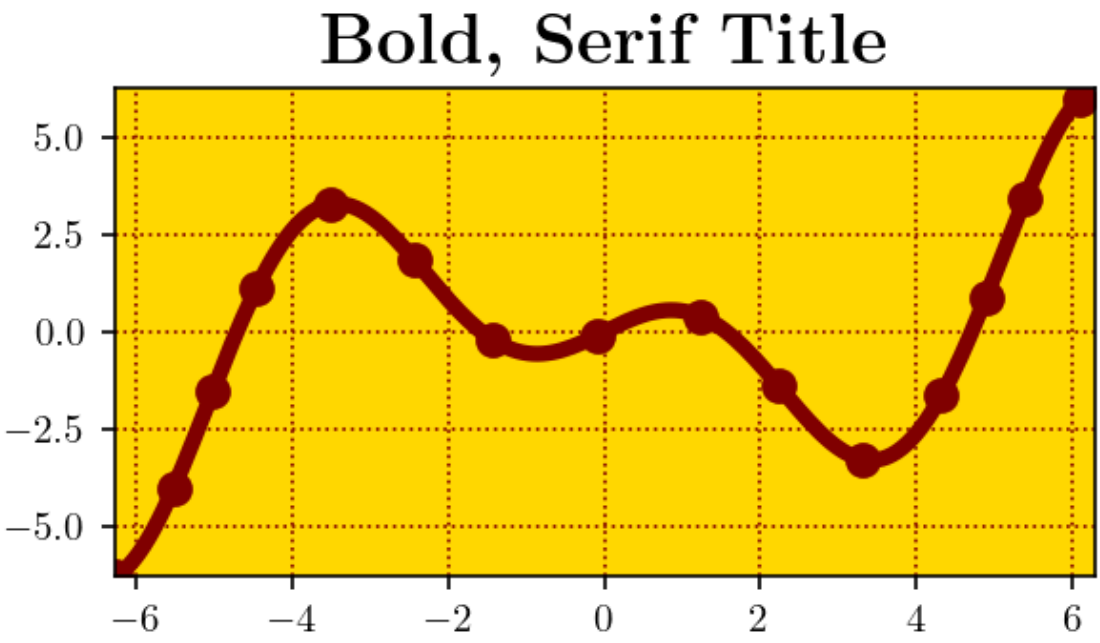
The `ax.set()` function can be used to set the limits at the same time as the aspect ratio is set to `"equal"`. Use the keyword arguments `xlim=` and `ylim=` and assign a tuple containing the lower and upper limits to them.

```
fig, ax = plt.subplots(figsize=(4, 3), dpi=150)
ax.plot(x, y1, c='green', lw=2, ls=':')
limits=(-2*np.pi, 2*np.pi)
ax.set(title="Plot Title", xlabel="x-axis", ylabel="y-axis")
# make the axes fit the data
ax.set(aspect="equal", xlim=limits, ylim=limits)
plt.show()
```



The following plot uses sets the aspect to 0.5; making the size of the y axis units half that of the x axis units. It also adds markers for some of the data points by adding `markevery=0.1` as an argument in the `plot()` function. This will place a marker every 10% of the way along the curve. The markers are also made larger by setting with `markersize=8` in the `plot()` function.

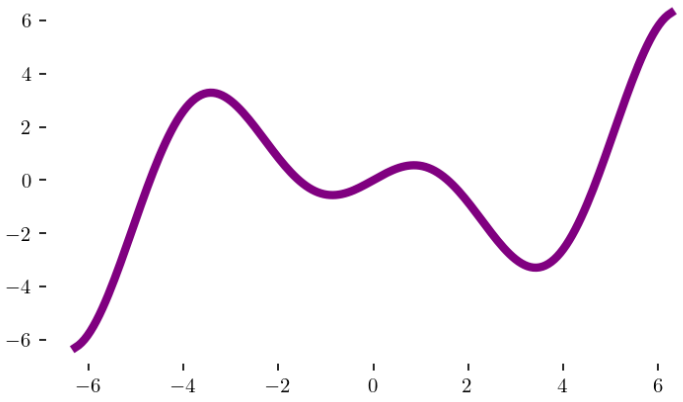
```
plt.rcParams['font.family'] = 'serif'
rc('text', usetex=True)
fig, ax = plt.subplots(figsize=(4,3), dpi=150)
ax.plot(x, y1, 'o-', c='maroon', lw=4, markevery=0.1, markersize=8)
# circle markers with solid line ('o-'), bigger markers and
# place markers every 0.1 (10%) along x-axis

ax.axis([-2*np.pi, 2*np.pi, -2*np.pi, 2*np.pi])
ax.set(aspect=0.5)
ax.set_facecolor('gold')
ax.set_title(r'$\textrm{\bf Bold, Serif Title}$', fontsize=18)
ax.grid(c='maroon', ls=':')
plt.show()
```



5.6 Frames and Tick Mark Visiblility

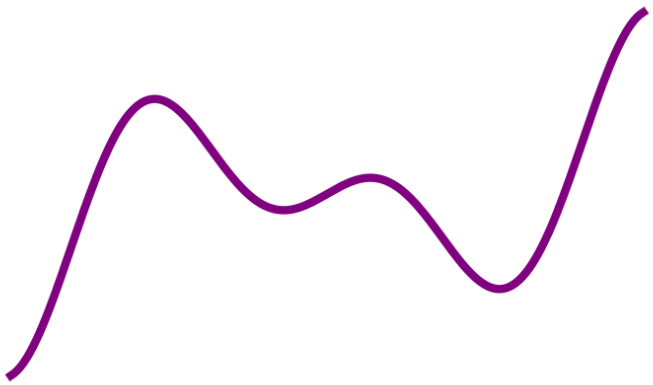
It is possible to turn off the "frame" around the edge of a plot axes by setting `ax.set_frame_on()` to `False`. The tick marks and numbers still remain in this case, as only the frame was removed.

```
fig, ax = plt.subplots(figsize=(5,3), dpi=150)
ax.plot(x, y1, lw=4, c='purple')
ax.set_frame_on(False) # turn off (or on) box around axes
plt.show()
```



Only the plot curve itself will remain if the frame and the entire axis is turned off. Do so by calling the `ax.set_axis_off()` function.

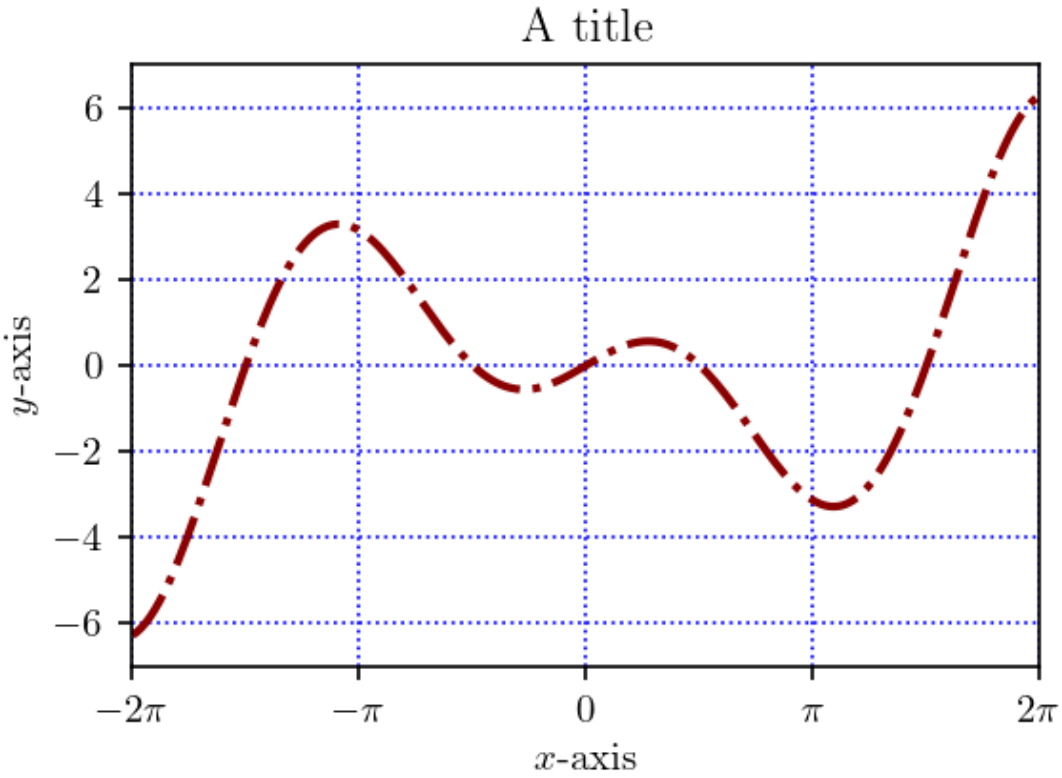
```
fig, ax = plt.subplots(figsize=(5,3), dpi=150)
ax.plot(x, y1, lw=4, c='purple')
ax.set_axis_off()
plt.show()
```



5.7 Fancy Text and Symbols

Turning on TEX for fancy formatting of text objects and setting the font family to `'serif'` will add a touch of "class" to a plot. Notice that the tick mark labels use strings like `'$-2\pi$'`. Sets of dollar signs are placed around "fancy" symbols and calculations that should be displayed like they would be in a math book. Here, `\pi` will be turned into the actual π symbol. Placing dollar signs around text will turn them italic so they look like variables instead of standard text. Some helpful TEX commands include `\pi`, `\sin`, `\cos`, `\tan`, `^2` (squared), and `_2` (subscript). It may be necessary on some occasions to use a 'raw' string by adding `r` before the quotation mark.

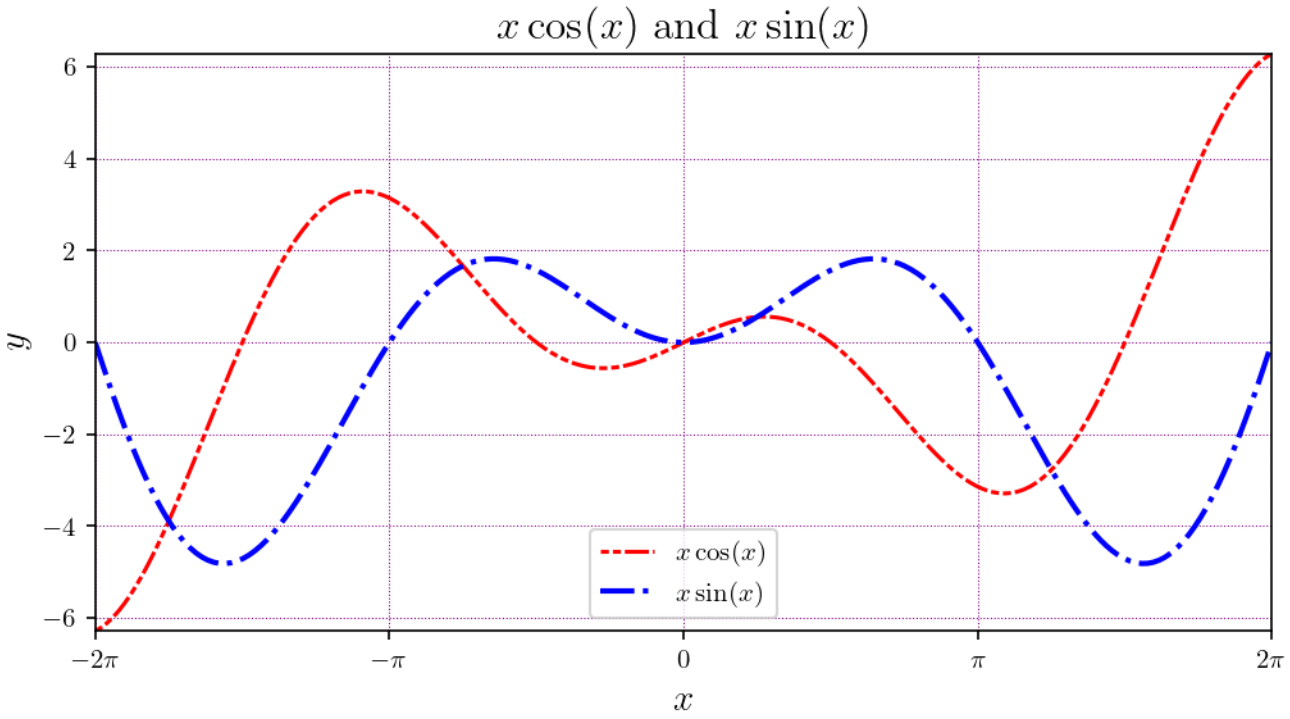
```
plt.rcParams['font.family'] = 'serif'
rc('text', usetex=True)
fig, ax = plt.subplots(figsize=(4,3), dpi=150)
ax.plot(x, y1, lw=2, c='darkred', ls='-.')
ax.set(title='A title', xlabel='$x$-axis', ylabel='$y$-axis')
ax.set_xticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi])
ax.set_xticklabels(['$-2\pi$', '$-\pi$', '$0$', '$\pi$', '$2\pi$'])
ax.set_xlim(-2*np.pi, 2*np.pi)
ax.set_ylim(-7, 7)
ax.grid(True, ls=':', c='blue')
plt.show()
```



5.8 Multiple Plots in the Same Axes

Two plotted curves can coexist in the same axes object by using two `plot()` functions in the same axes. The first plot has also added a custom line style (look up how on the *Mathplotlib* website). Fancy text and symbols are used in various places. A legend has been added by first including the `label=str` keyword argument to each `plot()` function call and then adding the `ax.legend()` function call. By default the legend will be placed where it best fits. Including the `loc='lower center'` keyword argument will center the legend near the bottom; some other options are `'upper right'`, `'upper left'`, `'lower right'`, `'lower left'`.

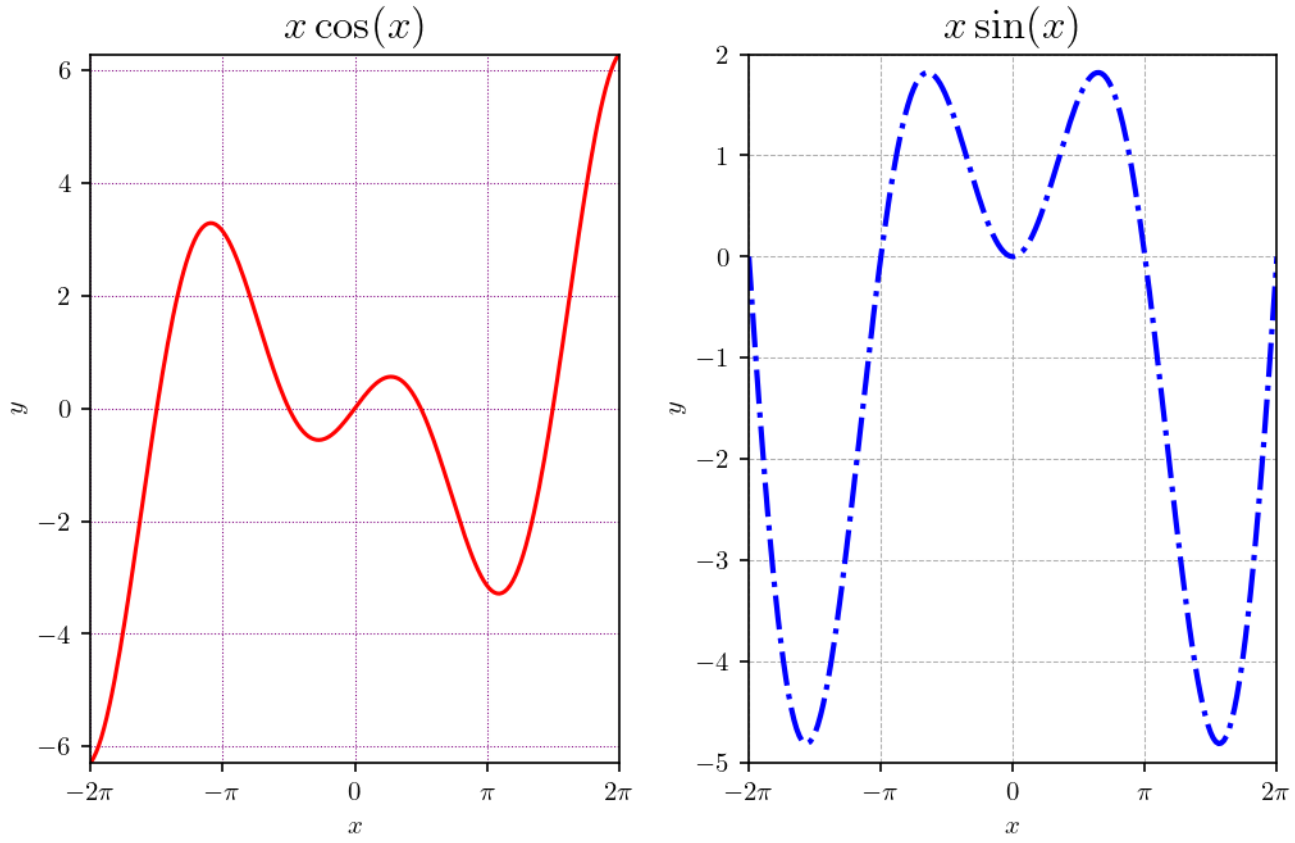
```
fig, ax = plt.subplots(figsize=(7,4), dpi=150)
ax.plot(x, y1, lw=1.5, c='red', ls=(0,(2,1,2,1,6,1)), label='$x\cos(x)$')
ax.plot(x, y2, lw=2, c='blue', ls='-.', label='$x\sin(x)$')
ax.set_xlabel('$x$', fontsize=14)
ax.set_ylabel('$y$', fontsize=14)
ax.set_title(r'$x\cos(x)$ and $x\sin(x)$', fontsize=16)
ax.grid(True, linestyle=':', lw=0.5, c='purple')
ax.set_xticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi])
ax.set_xticklabels(['$-2\pi$', '$-\pi$', '$0$', '$\pi$', '$2\pi$'])
ax.set_xlim([-2*np.pi, 2*np.pi])
ax.set_ylim([-2*np.pi, 2*np.pi])
ax.legend(loc='lower center') # legend at the lower center area of 'ax'
plt.show()
```



5.9 Multiple "Subplots" – Side-by-Side

Multiple plots can be placed side-by-side or above-below each other (or even in grids) in the same figure by adding some arguments to the `plt.subplots()` function. Adding two integer arguments immediately after the opening parentheses will set the number of row and columns to use for the subplots. The keyword arguments `nrows=r` and `ncols=c` could also be used instead. If the product of the rows and columns values is not one, then the `plt.subplots()` function will return a tuple of axes objects. This means that a tuple of axes names should be used on the left side of the equal sign. The following code creates side-by-side subplots with the axes names `ax1` and `ax2`. Use `plt.tight_layout()` before using `plt.show()` any time multiple subplots are created.

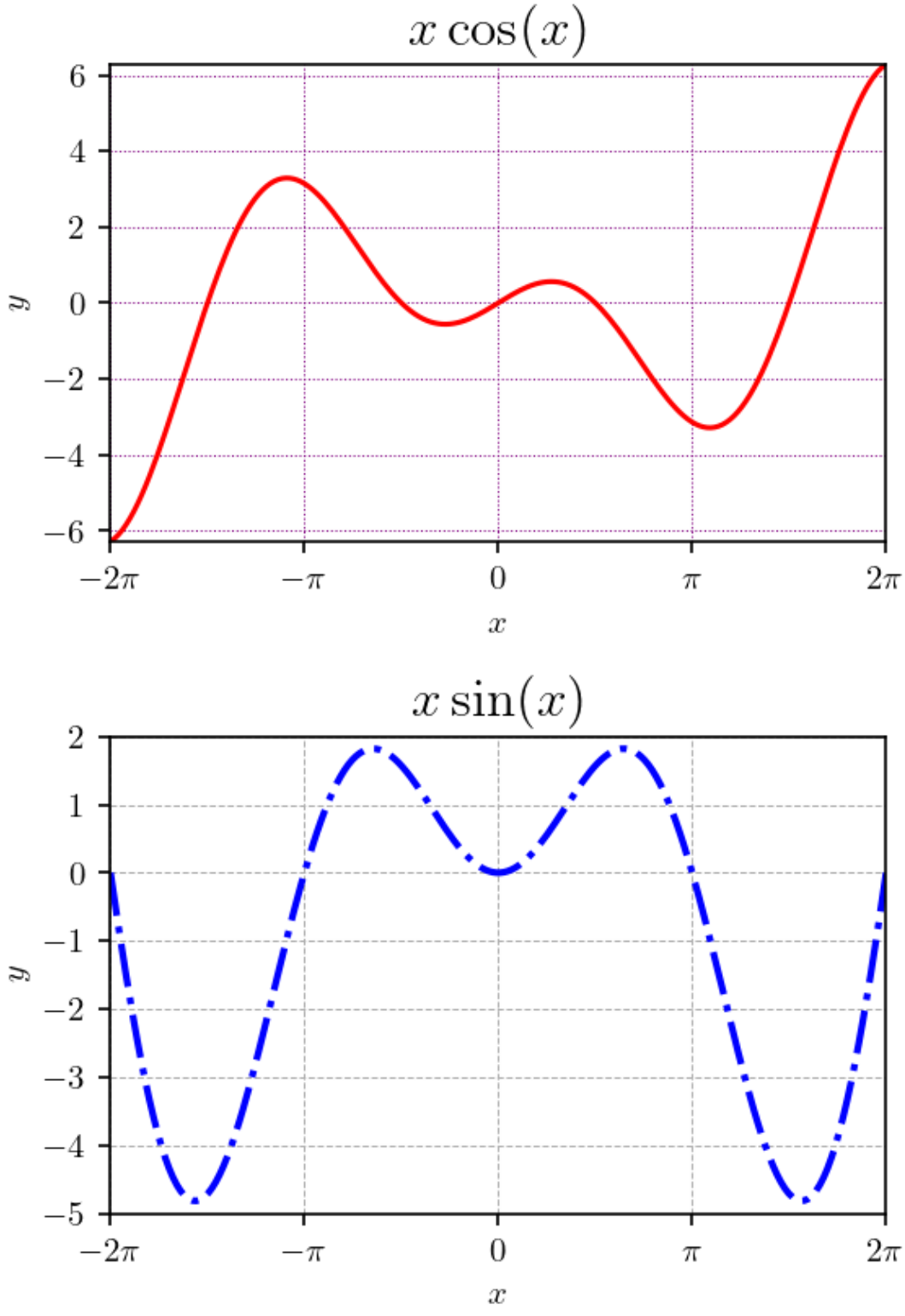
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7,5), dpi=150)
ax1.plot(x, y1, lw=1.5, c='red', ls='-')   # plotting to the left axes
ax2.plot(x, y2, lw=2, c='blue', ls='-.')   # plotting to the right axes
ax1.set_xlabel(r'$x$')
ax2.set_xlabel(r'$x$')
ax1.set_ylabel(r'$y$')
ax2.set_ylabel(r'$y$')
ax1.set_title(r'$x\cos(x)$', fontsize=16)
ax2.set_title(r'$x\sin(x)$', fontsize=16)
ax1.grid(True, linestyle=':', lw=0.5, c='purple')
ax2.grid(True, linestyle='--', lw=0.5)
ax1.set_xticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi])
ax1.set_xticklabels([r'$-2\pi$', r'$-\pi$', r'$0$', r'$\pi$', r'$2\pi$'])
ax2.set_xticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi])
ax2.set_xticklabels([r'$-2\pi$', r'$-\pi$', r'$0$', r'$\pi$', r'$2\pi$'])
ax1.set_xlim([-2*np.pi, 2*np.pi])
ax2.set_xlim([-2*np.pi, 2*np.pi])
ax1.set_ylim([-2*np.pi, 2*np.pi])
ax2.set_ylim([-5, 2])

# 'suptitle' is the title on top of the figure, not a specific axes
fig.suptitle(r'$\textrm{\bf Title for both plots}$', fontsize=18)

#using 'tight_layout()' on the figure gives nicer results for subplots
fig.tight_layout()
plt.show()
```


Title for both plots
$x\cos(x)$
$x\sin(x)$
y
x
-2π
$-\pi$
0
π
2π
6
4
2
0
−2
−4
−6
1
−1
−3
−5

5.10 Multiple "Subplots" – Stacked

Setting the number of rows to 2 and columns to 1 for the previous plot will stack them one on top of the other.

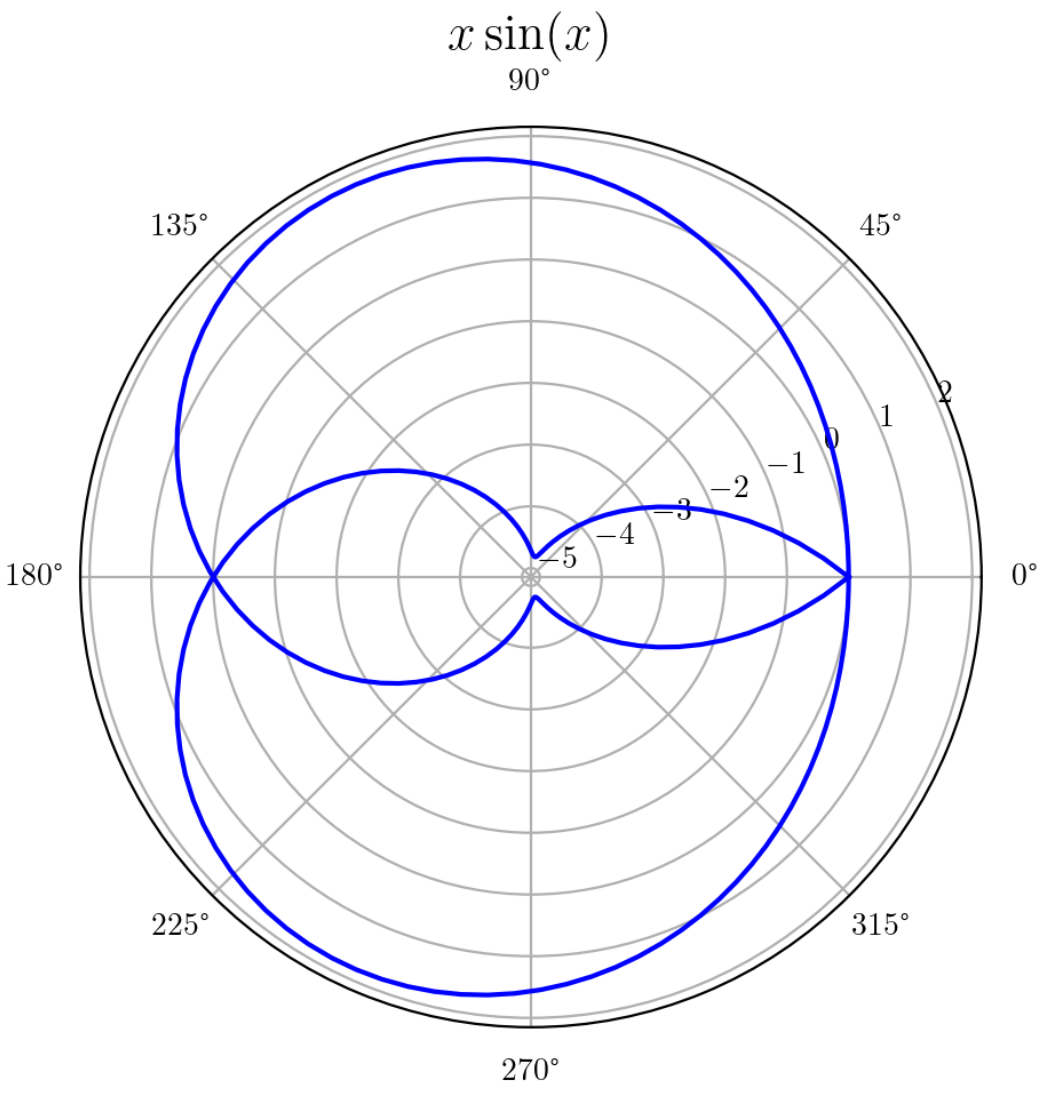
```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(4, 6), dpi=150)
ax1.plot(x, y1, lw=1.5, c='red', ls='-')   # plotting to the left axes
ax2.plot(x, y2, lw=2, c='blue', ls='-.')   # plotting to the right axes
ax1.set_xlabel(r'$x$')
ax2.set_xlabel(r'$x$')
ax1.set_ylabel(r'$y$')
ax2.set_ylabel(r'$y$')
ax1.set_title(r'$x\cos(x)$', fontsize=16)
ax2.set_title(r'$x\sin(x)$', fontsize=16)
ax1.grid(True, linestyle=':', lw=0.5, c='purple')
ax2.grid(True, linestyle='--', lw=0.5)
ax1.set_xticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi])
ax1.set_xticklabels([r'$-2\pi$', r'$-\pi$', r'$0$', r'$\pi$', r'$2\pi$'])
ax2.set_xticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi])
ax2.set_xticklabels([r'$-2\pi$', r'$-\pi$', r'$0$', r'$\pi$', r'$2\pi$'])
ax1.set_xlim([-2*np.pi, 2*np.pi])
ax2.set_xlim([-2*np.pi, 2*np.pi])
ax1.set_ylim([-2*np.pi, 2*np.pi])
ax2.set_ylim([-5, 2])

fig.suptitle(r'$\textrm{\bf Title for both plots}$', fontsize=16)

fig.tight_layout()
plt.show()
```


Title for both plots
$x\cos(x)$
6
4
2
0
−2
−4
−6
y
-2π
$-\pi$
0
π
2π
x
$x\sin(x)$
2
1
0
−1
−2
−3
−4
−5
y
-2π
$-\pi$
0
π
2π
x

6 Polar Plots

Polar plots use the same `plot()` function as x, y plots except the sequences used are the angle in radians θ and the radius r (in that order). However, the axes environment needs to be set up for a polar plot before using `plot()`. When using `plt.subplots()` the keyword argument `subplot_kw={'polar':True}` needs to be included.

```
fig, ax = plt.subplots(figsize=(5, 5), dpi=200,
                       subplot_kw={'polar':True})
# then just use 'plot()' as usual except (theta, r) not (x, y)
ax.plot(x, y2, 'b', lw=1.5)
ax.set_title('$x\sin(x)$', fontsize=16)
plt.show()
```



The plot below adds a second polar in the same figure by setting the `plt.subplots()` row and column options to 1 and 2, respectively. Also, the added plot (on the left) has the y tick marks set to intervals of π. The radial r axis is equivalent to the y axis for polar plots. Notice that the x (θ) axis ticks for the `ax1` (left) axes are set to an empty list, so there are no angle markings on that plot.

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8,4),
                               subplot_kw={'polar':True})
# then just use 'plot()' as usual except (theta, r) not (x, y)

# 'ax1' uses left subplot
ax1.plot(x, y1, 'r', lw=1.5)
ax1.set_title('$x\cos(x)$', fontsize=16)
ax1.set_rlim(-2*np.pi, 2*np.pi)
ax1.set_xticks([])
ax1.set_yticks([-2*np.pi, -np.pi, 0, np.pi, 2*np.pi])
ax1.set_yticklabels(['$-2\pi$', '$-\pi$', '$0$', '$\pi$', '$2\pi$'],
                    fontsize=12)

# 'ax2' uses right subplot
ax2.plot(x, y2, 'b', lw=1.5)
ax2.set_title('$x\sin(x)$', fontsize=16)
fig.tight_layout()
plt.show()
```